Environments

- Environments map names to values.
- They consist of chains of environment frames.
- An environment is either a global frame or a first (local) frame chained to a parent environment (which is itself either a global frame or . . .).
- We say that a name is bound to a value in a frame.
- The value (or meaning) of a name in an environment is the value it is bound to in the first frame, if there is one, . . . or if not, the meaning of the name in the parent environment (recursively).

Summary: Environments

- Every expression and statement is evaluated (executed) in an environment, which determines the meaning of its names.
- Expressions and subexpressions (pieces of an expression) are evaluated in the same environment as the statement or expression containing them.
- Assigning to a variable binds a value to it in (for now) the first frame of the environment in which the assignment is executed.
- Def statements bind a name to a function value in the first frame of the environment in which the def statement is executed.
- Calling a user-defined function creates a new local environment frame that binds the function's formal parameters to the operand values (actual parameters) in the call.
- This new local frame is attached to an existing (parent) frame that is taken from the function value that is called, forming a new local chain (of environment frames) in the call.
- Defining a function binds a name to a function value in the first frame of the environment in which the def statement is executed.
- Every expression and statement is evaluated (executed) in an environment, which determines the meaning of its names.

Example: Evaluation of a Call: `sum(square(3,4))`

- `square`: . . .
- `mul`, `abs`. . .
- `sum`

- `square`
- `func square(x)[parent=Global]`
- `func sum square(x, y)[parent=Global]`
- `sum square(3, 4)`
- `25`
- `square(x) = x*x`
- `square(3) = 9`
- `square(4) = 16`
- `x*x`

execute this

What Does This Do (And Why)?

```
def id(x):
    return x
print(id(id)(id(13)))
```
The question of a value…

Answer

def id(x):
    return x

print(id(id)(id(13)))

• We'll denote the user-defined function value created by `def id:` by the shorthand `id`.
  • Evaluation proceeds like this:
    
    `id(id)(id(13))`  
    =  
    `id(id)(13)` (because first `id` call returns its argument).
    =  
    `id(13)` (because inner `id` call returns its argument).
    =  
    13 (because call to returned `id` value returns its argument).

• Important: There is nothing new on this slide! Everything follows from what you've seen so far.

Recap

• Every expression or statement is evaluated in an environment—a sequence of frames.
  • Every frame (except the global frame) is linked to a parent frame.
  • Every function value is linked to the environment in which its definition is evaluated.
  • Every function call creates a new local frame that is linked to the same frame as the function value being called.
  • The total effect is the same as for the substitution model, but we can also handle changes in the values of variables.
  • Looking ahead, there are still two constructs—`global` and `nonlocal`—that will require additions.

Control

• The expressions we've seen evaluate all of their operands in the order written.
  • While there are very clever ways to do everything with just this (challenge!), it's generally clearer to introduce constructs that control the order in which their components execute.
  • A control expression evaluates some or all of its operands depending on the kind of expression, and typically on the values of its operands in order to introduce constructs that can do clever things.
  • A statement is a construct that produces no value, but is used solely for its side effects.
  • A control statement is a statement that, like a control expression, evaluates some or all of its operands, etc.
  • We typically speak of statements being executed rather than evaluated, but the two concepts are essentially the same, apart from the question of a value.

So how does this work with environments?

Recap

In lecture #2, I had this example:

```python
def incr(n):
    def f(x):
        return n + x
    return f

incr(5)(6)
```

• We evaluated the argument to `print` by substitution:
  
  `incr(5) ===>
  def f(x):
      return 5 + x
  return f ===>
  λ x: 5 + x`

  `incr(5)(6) ===> (λ x: 5 + x)(6) ===> 5 + 6 ===> 11`

• So how does this work with environments?

Environments for incr (I)

```python
def incr(n):
    def f(x):
        return n + x
    return f
```

# Break incr(5)(6)

```python
g = incr(5)
print(g(6))
```

- Global `incr`:
  - Parent: `Global`
  - `g`: `func incr(n)[parent=Global]`
- Evaluation body of `incr` here:

- Global `incr` is evaluated in the global environment.
- The parent points of `incr` is `Global` because the definition of `incr` was evaluated in the global environment.
- The parent pointer for the value of `g` (returned by `incr(5)`) is `f1`, not `Global`, because the definition of `f` was evaluated in `f1`.

Environments for incr (II)

```python
g = incr(5)
print(g(6))
```

- Evaluation of `g(6)` here:
  - `f2` gets its parent pointer from `g`'s value, since it is the local frame for evaluating a call to `g`.
  - Same rule for `f1`.

Recap

• Because the definition of `g` is evaluated in `f1`, `g` is defined in the global environment. The parent points of `incr` is `Global` because the definition of `incr` was evaluated in the global environment.

- Global ` incr`
  - `g`: `func incr(n)[parent=Global]`
- Evaluation body of `g` (i.e., `f`) here:

- `f2` gets its parent pointer from `g`'s value, since it is the local frame for evaluating a call to `g`. (Same rule for `f1`.)

Answer

- If `g` were defined outside of `incr`, then the parent points of `g` would be `Global`, not `f1`.

- By the substitution model, the value created by `def id()` is:
  
  `((id)(id))(id)`
Conditional Expressions (I)

• The most common kind of control is conditional evaluation (execution).

• In Python, to evaluate:

\[
\text{TruePart if Condition else FalsePart}
\]

– First evaluate Condition.
– If the result is a "true value", evaluate TruePart; its value is then the value of the whole expression.
– Otherwise, evaluate FalsePart; its value is then the value of the whole expression.

• Example:

If \( x \) is 2:

\[
\frac{1}{x} \text{ if } x \neq 0 \text{ else } 1
\]

\[
\frac{1}{x} \text{ if } 2 \neq 0 \text{ else } 1
\]

\[
\frac{1}{x} \text{ if True else } 1
\]

\[
\frac{1}{x}
\]

\[
\frac{1}{2}
\]

\[
0.5
\]

If \( x \) is 0:

\[
\frac{1}{x} \text{ if } x \neq 0 \text{ else } 1
\]

\[
\frac{1}{x} \text{ if } 0 \neq 0 \text{ else } 1
\]

\[
\frac{1}{x} \text{ if False else } 1
\]

\[
1
\]

• Last modified: Sat Feb 18 22:56:06 2017 CS61A: Lecture #3

Conditional Expressions (II)

• To evaluate Left and Right:

– Evaluate Left.
– If it is a false value, that becomes the value of the whole expression.
– Otherwise the value of the expression is that of Right.

• This is an example of something called "short-circuit evaluation".

• For example,

\[
5 \text{ and } \text{"Hello"}
\]

\[
\text{"Hello"}
\]

\[
\text{[] and } 1 / 0
\]

\[
\text{[]}
\]

\[
(1/0 \text{ is not evaluated.})
\]

• Another example of short-circuit evaluation.

• Conditions in conditional constructs can have any value, not just True or False.

• For convenience, Python treats a number of values as indicating "false":

– False
– None
– 0
– Empty strings, sets, lists, and dictionaries.

• All else is a "true value" by default.

• Empty strings, sets, lists, tuples, and dictionaries.

• For convenience, Python treats a number of values as indicating False.

• Conditions in conditional constructs can have any value, not just True.

• For example:

\[
1 \text{ if } 0 \text{ else } 5
\]

\[
1 \text{ if } [] \text{ else } 5
\]

Example: If \( x \) is 2:

\[
\text{if } x > 0 \text{ then } 1 \text{ else if } x == 0 \text{ then } 0 \text{ else } -1
\]

\[
\text{if } x > 0 \text{ else if } x == 0 \text{ else } -1
\]

• Last modified: Sat Feb 18 22:56:06 2017 CS61A: Lecture #3

Conditional Expressions (III)

• To evaluate Left or Right:

– Evaluate Left.
– If it is a true value, that becomes the value of the whole expression.
– Otherwise the value of the expression is that of Right.

• Another example of short-circuit evaluation.

• For example,

\[
5 \text{ or } \text{"Hello"}
\]

\[
5
\]

\[
\text{[]} \text{ or } \text{"Hello"}
\]

\[
\text{"Hello"}
\]

\[
\text{[]} \text{ or } 1 / 0
\]

\[
?
\]

• Last modified: Sat Feb 18 22:56:06 2017 CS61A: Lecture #3

Conditional Statement

• Finally, this all comes in statement form:

\[
\text{if } \text{Condition}1:\n\text{Statements}1\#
\text{Indented blocks are called suites.}
\ldots
\# They group statements.
\text{elif } \text{Condition}2:\n\text{Statements}2
\ldots
\text{else }:\n\text{Statements}n
\ldots
\]

• Execute (only) Statements1 if Condition1 evaluates to a true value.
• Otherwise execute Statements2 if Condition2 evaluates to a true value (optional part).
• . . .
• Otherwise execute Statementsn (optional part).

• Last modified: Sat Feb 18 22:56:06 2017 CS61A: Lecture #3